NASA CR-144989

ANOPP PROGRAMMING AND DOCUMENTATION STANDARDS DOCUMENT

Prepared under Contract No. NAS1-13983 by
CONTROL DATA CORPORATION
3217 N. Armistead Avenue
Hampton, VA 23690

for

NATIONAL AERONAUTICS & SPACE ADMINISTRATION

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1.1 SCOPE

These standards define the requirements for preparing software for the Aircraft Noise Prediction Program (ANOPP). It is the intent of these standards to provide definition, design, coding, and documentation criteria for the achievement of a unity among ANOPP products.

These standards apply to all of ANOPP's standard software system. The standards as expressed in this publication encompass philosophy as well as techniques and conventions.

1.2 DESIGN

1.2.1 Module Standards

The Aircraft Noise Prediction Program will utilize the concepts of "composite design" for program structure. Composite design involves the construction of a program in terms of modular structure and module interfaces.

A module is a group of program statements that can receive input data, perform one or more transformations on that data, and return output data. The modules for ANOPP will have the following general characteristics.

a. The executable and comment statements for the module can be listed contiguously.

b. The statements are enclosed by identifiable boundaries; e.g., in FORTRAN, from a PROGRAM or SUBROUTINE card to an END card.

c. The statements are considered to be a discrete and identifiable entity that can be referenced by the module name.

d. The module can be referenced from other parts of the program only by the module name or its single entry.

e. The module will have a single, common entry and a single, common exit.

f. The module can reference or CALL other modules, suspend its execution upon encountering the CALL statement and resume execution with the next immediate statement.

g. All called modules must return to their caller at the statement immediately following the call statement.

A module has three attributes: function, logic, and interfaces. For a composite design, a module should be described by its functions; i.e., what happens when the module is called. This characteristic can be described as data flow through the program. A functional description of a module should contain a verb, such as, "Find Largest Block". A module's logic describes the internal working or data flow within a module. Another attribute of a module, interconnection or interface, is concerned with module communication.

ANOPP modules will be designed to be as functionally independent as possible with minimum interface. Reliability, ease of understanding, and ease of maintenance are the objectives of these standards.

Guidelines to be followed to achieve the standards of modularity for ANOPP are:

a. Simplicity - Use the simplest solution, design and/or interface that is possible.

b. Design efficiency - Design a module to solve the current problem efficiently; i.e., never design a module to do more than it is required to do.

c. Aligned control and effect - Align modules and decisions in modules so modules that are directly affected by a decision are beneath and controlled by the module containing the decision.

d. High strength - Maximize binding, the relationships among the

elements of a module. For high strength or binding modules should:

1. have all elements related to the performance of a single function,
2. have a single entry and a single exit,
3. have a function which is easy to describe,
4. be independent from other modules,
5. be unsusceptible to errors from complex design and coding,
6. be usable by other programs,
7. and be modifiable without affecting other modules.

e. Low interconnection - Minimize coupling, the relationships between modules. To attain the desired low interconnection or loose coupling, modules should:

1. not directly reference other modules to:
   (a) modify a program statement,
   (b) refer to data in another module,
   (c) branch directly into another module, or
   (d) physically reside within another module,
2. not pass control information to other modules,
3. use only the following types of data interconnections:
   (a) argument lists,
   (b) common areas, and
   (c) data base members.

f. Size - Limit the size of a module to 100 lines of executable

source language statements. Clarity, simplicity and understanding are related to module size.

1.2.2 Depth of Design

Program structuring involves an analysis of the problem, the flow of data through the problem, and the transformations that occur on that data. Equally, it involves identification and definition of modules to solve the problem. The phases of program structuring involve:

a. definition of the structure of the problem,

b. identification of the input and output in the problem,

c. identification of the points of entry and exit for data,

d. reduction of the problem into a set of subordinate modules.

A graphic representation of a module and subordinate modules will result in a hierarchy of modules. The graphic representation is called a "hierarchy chart" and should be drawn to illustrate the problem's solution.

After the problem has been reduced into a set of subordinate modules, the process is repeated viewing each subordinate module as an independent problem that can be reduced into other subordinate modules. Some rules to follow are:

a. The entire structure should be constantly reviewed to take advantage of modules that are identical.

b. A module must be completely analyzed before its subordinates

can be analyzed.

c. The order in which modules at the same level are analyzed and reduced is not important. It is not necessary to analyze a module and all of its subordinates before starting another module.

Conditions for terminating this iterative reduction process are:

a. When a module can be reduced no further into independent functional modules.

b. When further reduction of a module leads to the undesirable attributes of low strength and high interconnections; i.e., low binding and high coupling.

c. When the logic of a module can be completely visualized; i.e., usually resulting in less than 100 executable statements.

d. When further reduction leads to highly specialized, unaligned, and inefficient sets.

e. When the resulting documentation (Chapin-style charts) can be drawn on two or less sheets of 8-1/2" x 11" paper. (One page is the desirable objective.)

The final design phase of a module should follow these steps:

a. A Chapin chart should be drawn to illustrate the module's logic structure.

b. Documentation should be set down to define the module's purpose, functions, inputs, and outputs. This constitutes the first portion of the module's prologue and is intended to be included with the source statement listing.

c. A "walk through" of the module should be staged by the design team to insure workability.

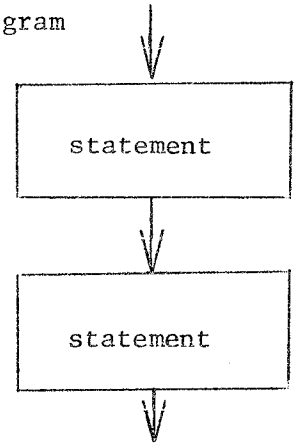
d. Pseudo code reflecting the logic structure outlined by the Chapin chart should be completed. This completes the module's prologue.

1.2.3 Logic Structures

All of the ANOPP modules will be logically designed so the execution flow will be sequential from one logic structure to the next logic structure. ANOPP module design will use four basic logic structures. Coding of a logic structure may require more than one executable source statement. No matter how complex the particular structure, upon completion of its execution, the next structure will be executed. This sequential flow logic is characteristic of "top-down" design. The four logic structures are defined with traditional graphics in the following paragraphs for educational value and comparison with the ANOPP standard Chapin charts.

1.2.3.1 Simple Sequence Structure

1.2.3.1.1 Logical Flow Diagram

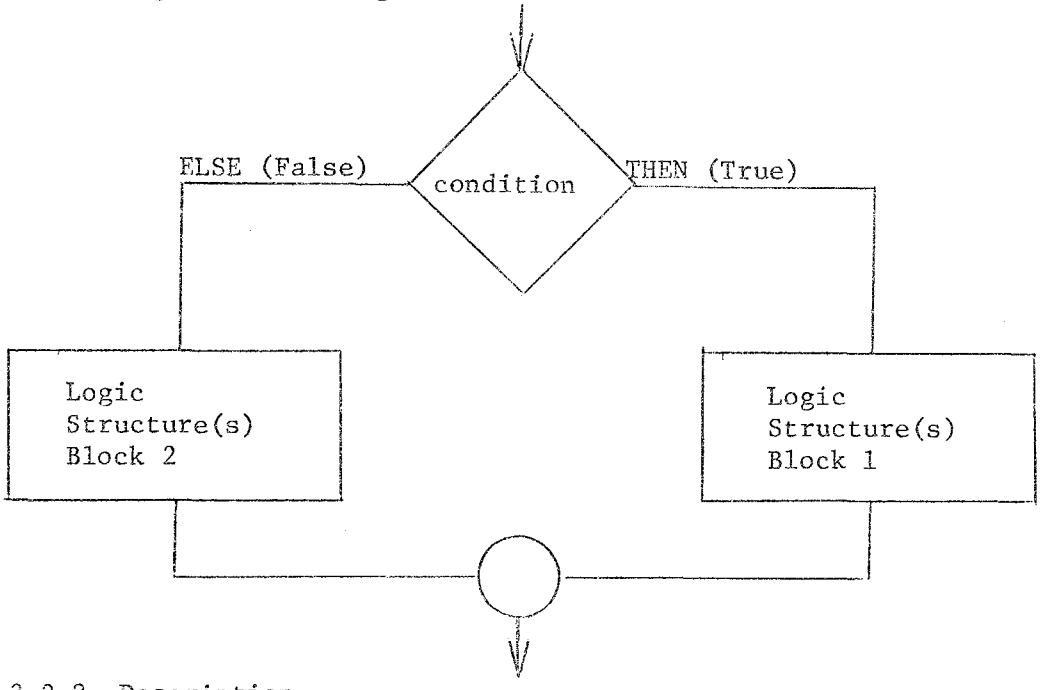


1.2.3.1.2 Description

Each statement within the structure is simple in that it performs one basic function; e.g., an assignment of an evaluated expression to a variable or a call to another module (or subordinate).

1.2.3.2 IF THEN/ELSE Structure

1.2.3.2.1 Logical Flow Diagram



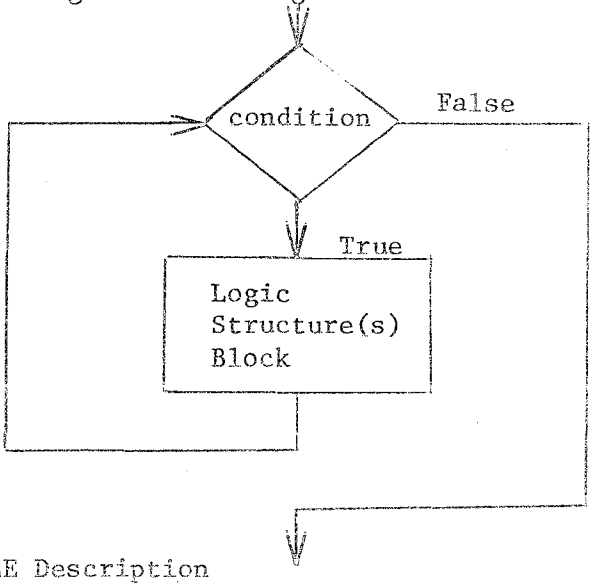
1.2.3.2.2 Description

The IF THEN/ELSE structure describes the flow sequence that occurs when there are two blocks of logic structure(s) and only one block should be executed according to a decision criteria or condition. The condition is a simple or a complex logical expression which has a value of True or False. If upon execution the condition is True,

the logic structure(s) of Block 1 (the THEN path) is(are) executed. If the condition is False, the logic structure(s) of Block 2 (the ELSE path) is(are) executed. When the last logic structure in the chosen path has been executed, control goes to the next logic structure or statement following the IF THEN/ELSE structure. This logic structure adheres to the "top-down" design in that the common entry is the point at which the condition is tested and the common exit leads to the next logic structure.

1.2.3.3 DO WHILE/DO UNTIL Structure

1.2.3.3.1 DO WHILE Logical Flow Diagram

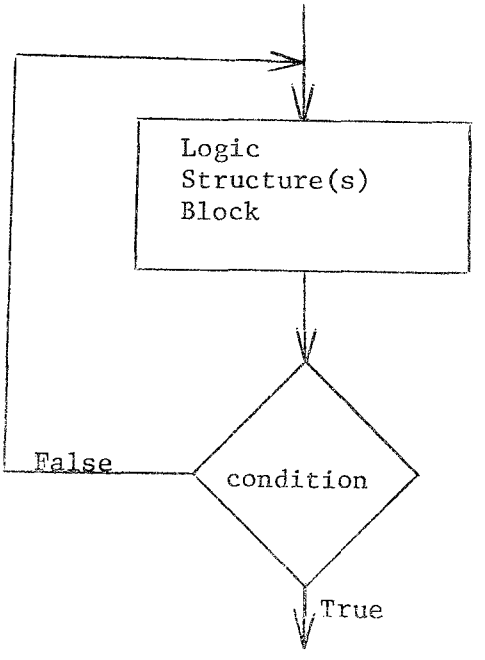


1.2.3.3.2 DO WHILE Description

A block of logic structure(s) which may or may not be executed one or more times depending on a given condition is described by the DO WHILE loop structure. The loop structure adheres to "top-down" design in that the loop is entered at one point and flow within the loop progresses

to one common exit point; i.e., the point at which the condition is tested. The condition is a simple or complex logical expression which has a value of True or False. The condition is tested at the beginning of the loop, before the execution of the logic structure(s) block, and if True the logic structure(s) is(are) executed. When execution of the logic structure(s) is(are) complete, control returns to the beginning of the loop and the condition is tested again. Looping continues until the condition is False; control then passes to the next logic structure following the DO WHILE structure.

1.2.3.3.3 DO UNTIL Logical Flow Diagram



1.2.3.3.4 DO UNTIL Description

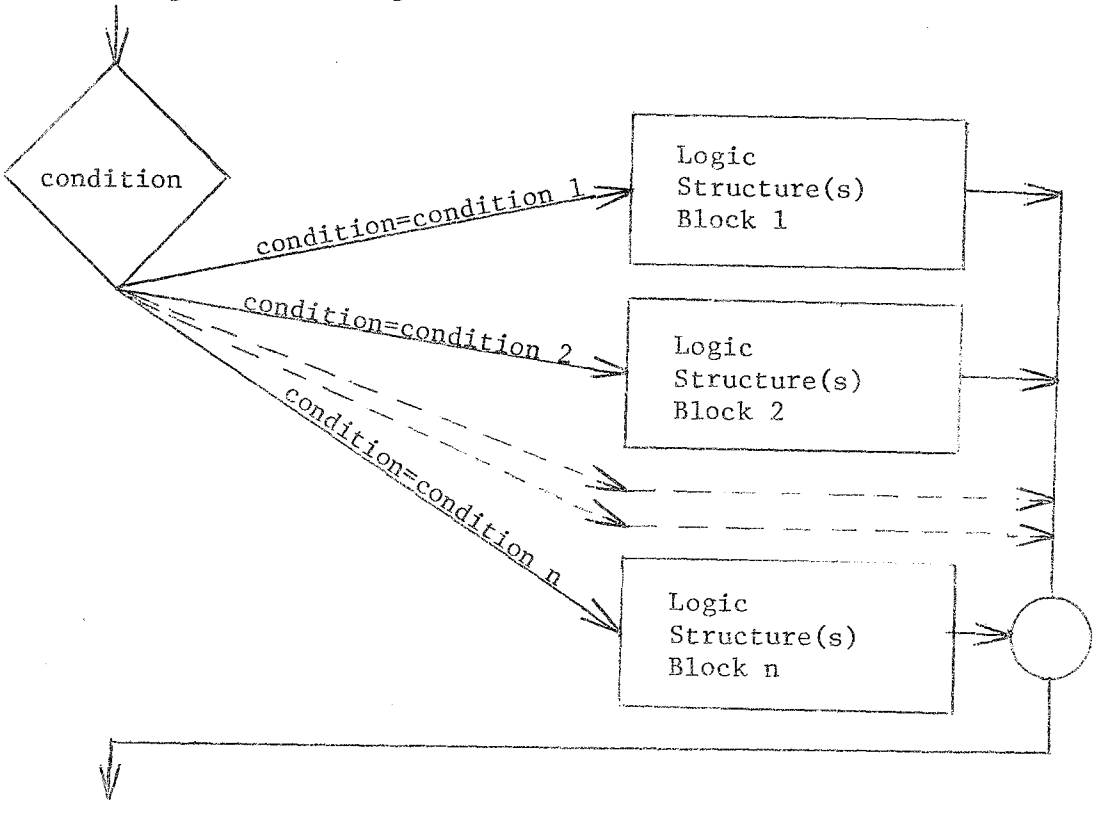
A block of logic structure(s) which will be executed one or more times depending on a given condition is described by the DO UNTIL

loop structure. The structure adheres to the "top-down" design in that the loop is entered at one point and flow within the loop progresses to one common exit point; i.e., the point at which the condition is tested.

The condition is a simple or a complex logical expression which has a value of True or False. The condition is tested at the end of the loop, after the execution of the logic sturcture(s) block, and if False the logic structure(s) block is executed again. This continues until the condition tested is True; control then passes to the next logic structure following the DO UNTIL structure.

1.2.3.4 CASE Structure

1.2.3.4.1 Logical Flow Diagram



1.2.3.4.2 Description

The CASE logic structure describes the flow sequence that develops when there are two or more blocks of logic structures, only one of which will be executed according to a given condition or decision criteria. The condition when evaluated must have a resulting value identical to one and only one of the conditions given (i.e., condition 1, condition 2, ..., condition n). Control will pass to the beginning of the logic structure's block identified by the matching condition. Upon completion of the chosen block, control will pass to the next logic structure following the CASE structure. The CASE structure adheres to the "top-down" design in that the structure is entered at one point, the test condition point, and after execution of the chosen block, control passes to one common exit and the next logic structure.

1.2.4 Design Documentation

The design phase should result in a description of the structure of the program with descriptions of the module and intermodule interfaces. For the ANOPP project, the design phase will result in (1) hierarchy charts of the areas of the program, (2) Chapin charts with external specifications (module prologue) for each module depicted on the hierarchy charts and (3) pseudo code.

1.2.4.1 Hierarchy Charts

A hierarchy chart will depict the results of the composite analysis

process (top-down reasoning - structural process). This creative process is necessary to arrive at the modular logic and involves the analysis of the problem, the flow of data through the problem, and subdivision of the problem into modules that will perform transformations on the data.

A hierarchy chart depicts each module, the level of the module (order), and the lines of communication for the module. In its optimal form, a heirarchy chart should be contained on one page (Figure 1). As an area can have several levels of modules, it may be necessary to place a module with its subsequent levels on additional pages; however, all subordinate modules on the same level should be placed on the same page. In the example illustrated in Figures 2 and 3, five levels of modules are necessary. The submodules of the module "Control to Next Level" (Level 3) could not be depicted on the same page and were subsequently placed on an additional page.

Each module will have a short title, descriptive of its function, and a name that is used to reference the module. Module names should be descriptive of the function.

Figure 4 is an example of a hierarchy chart depicting levels, short titles, and names.

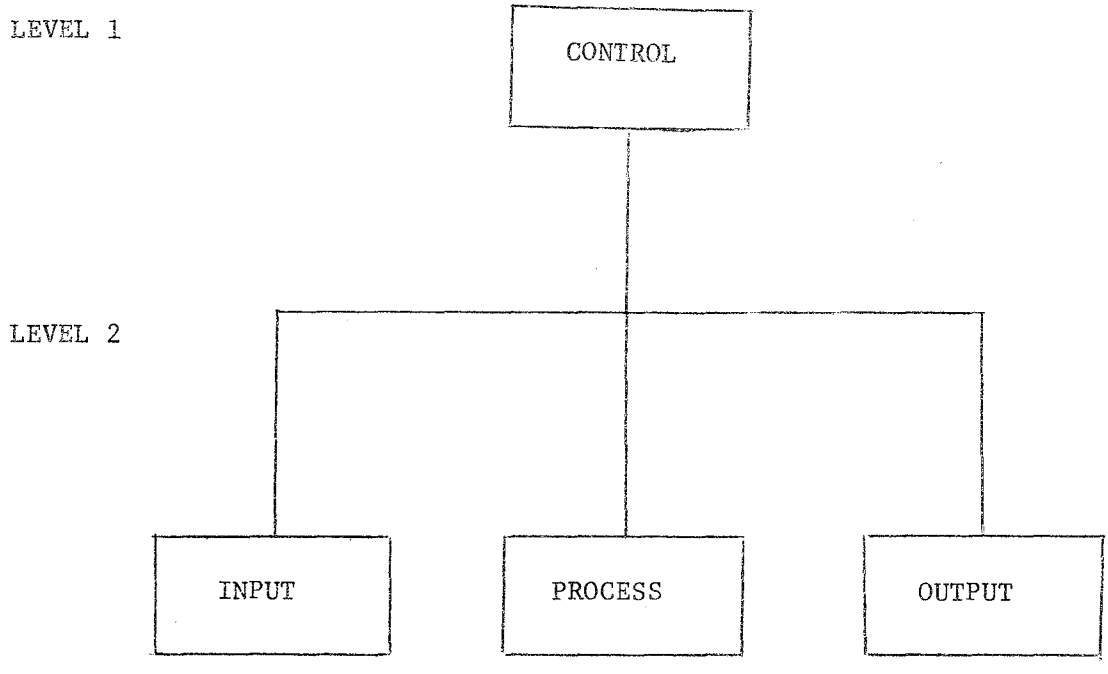


Figure 1. Hierarchy Chart: Basic Form

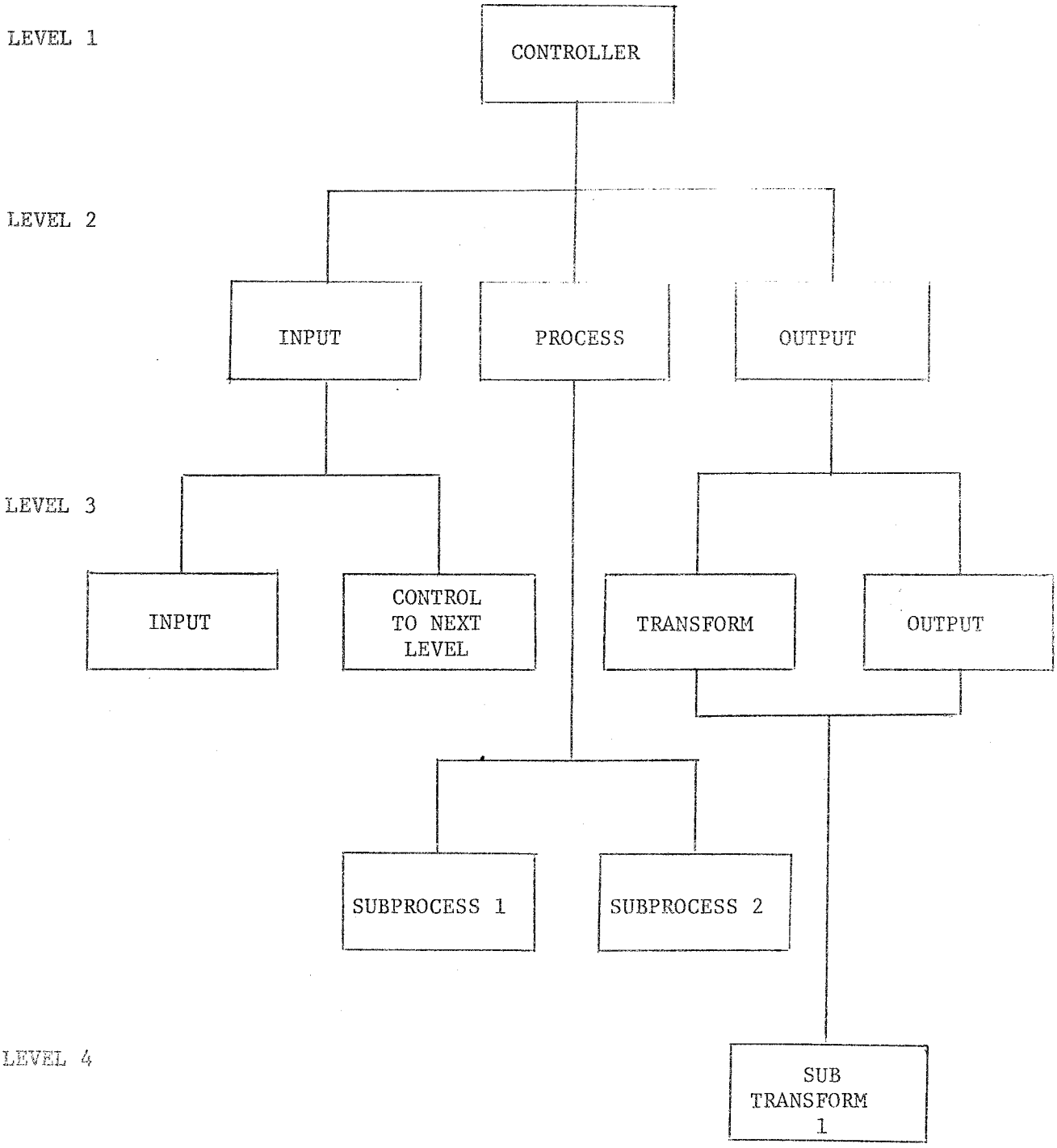


Figure 2. Hierarchy Chart: Basic Form

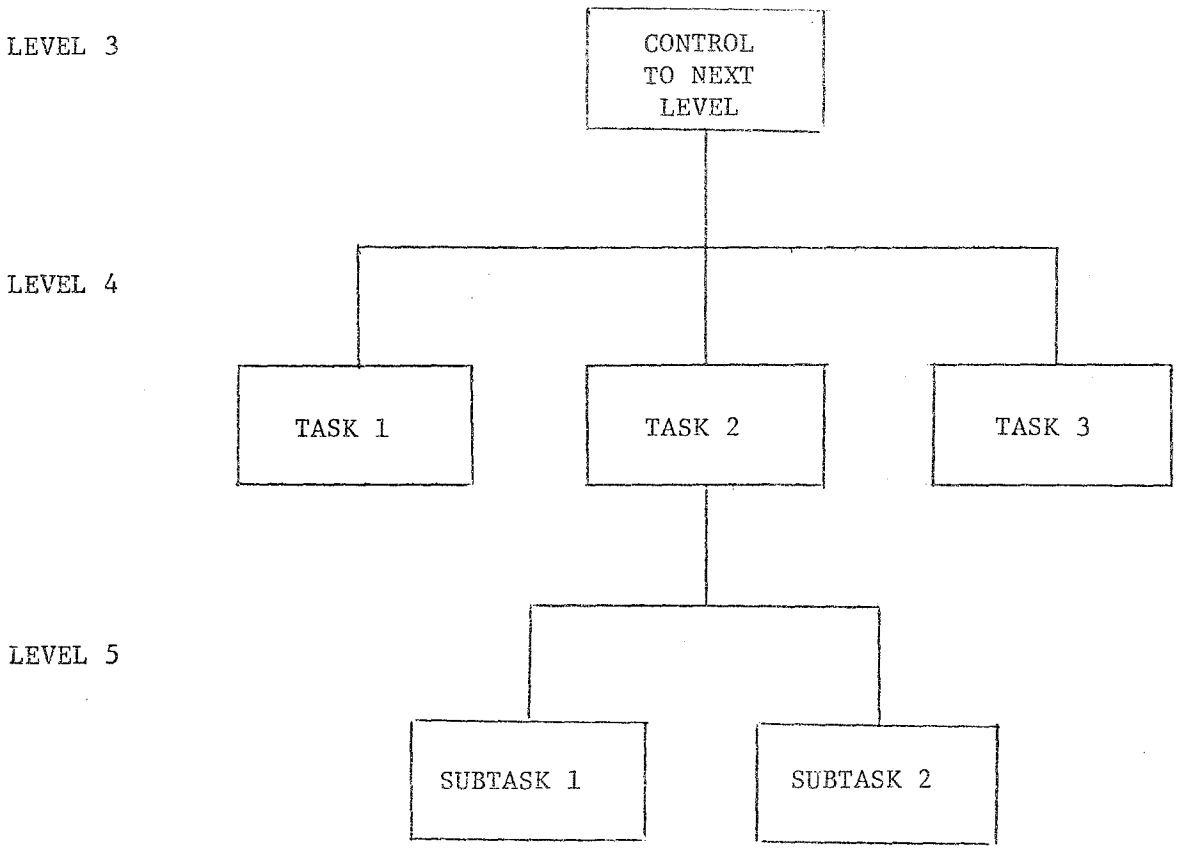


Figure 3. Hierarchy Chart: Basic Form

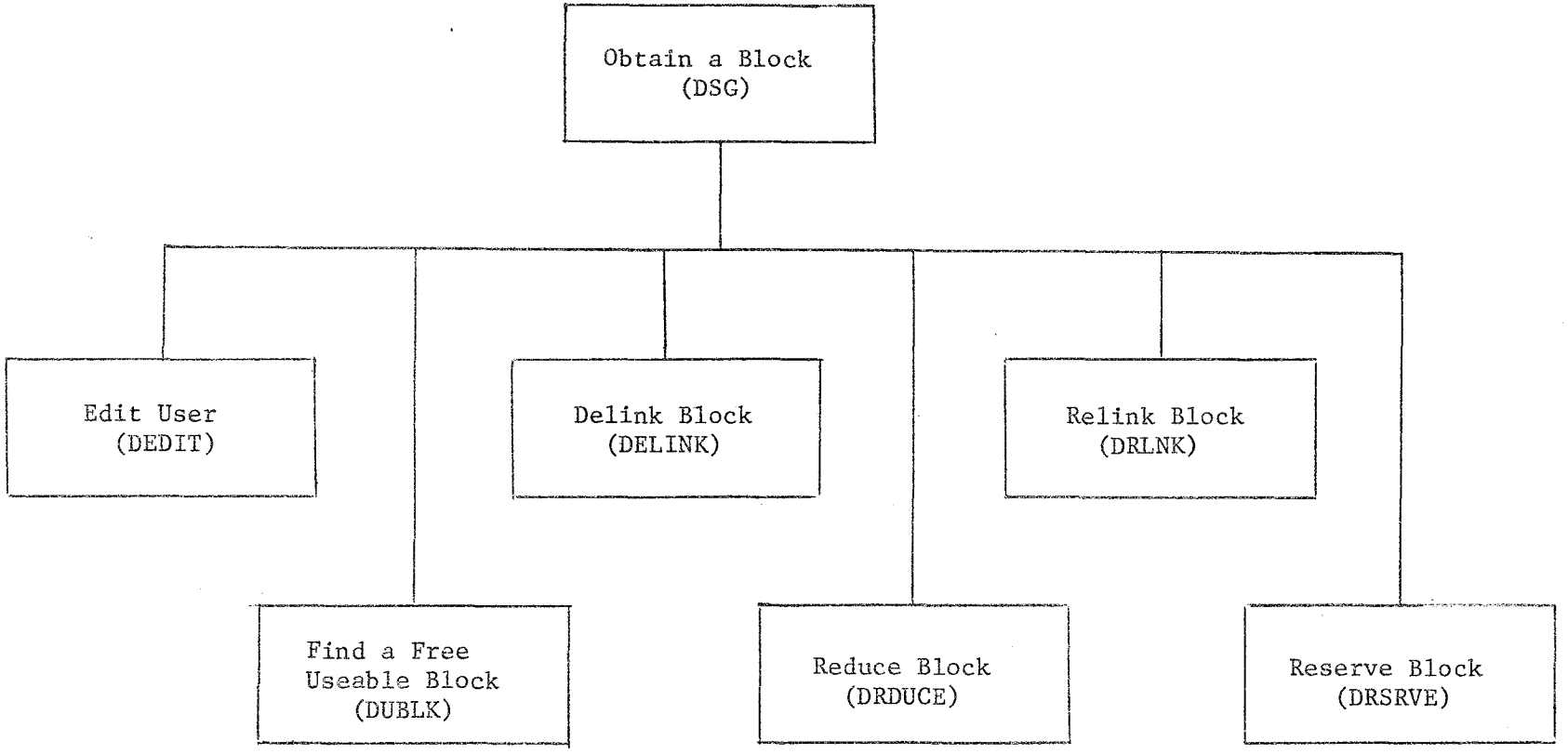


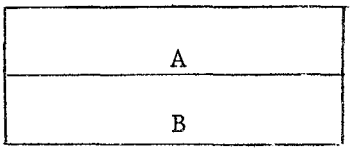
Figure 4. Hierarchy Chart

To obtain a block - Call DSG(USER,MIN,MAX,IDX,LENGTH,LERROR)

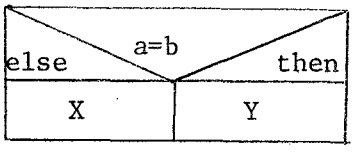
1.2.4.2 Chapin Charts

A Chapin chart will be drawn for each module depicted on the hierarchy chart. The drawn Chapin chart will detail the internal logic of the module using simple control structures. In a structured program module, any program function can be performed using one of four control structures. The Chapin forms for these structures are:

a. Simple sequence

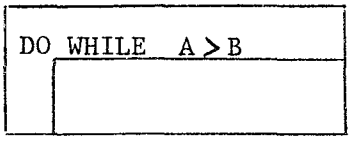


b. IF THEN/ELSE

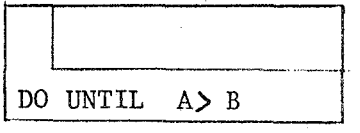


c. Repetition

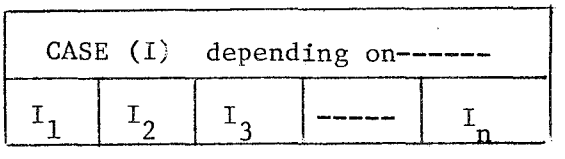
DO WHILE



DO UNTIL



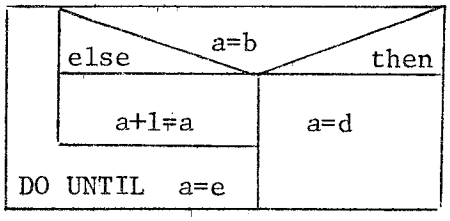
d. CASE

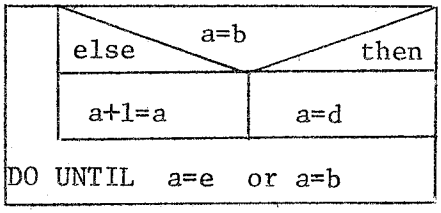


CASE I is really a generalization of the selection function (IF THEN/ELSE) from a two-valued to a multi-valued operation.

Any kind of processing, any combination of decisions, and any sort of logic can be accomodated with one of these control structures or a combination of these control structures. Each structure is characterized by a single point of transfer of control into the structure and a single point of transfer out of the structure. The control structures can be nested and still retain this characteristic.

A tricky situation, prevalent in current practice, arises when a designer desires to terminate a repetition block upon encountering a specific condition. If this termination is diagrammed as illustrated at the right, it violates the single entry/single exit principle of structured programming. However, equivalent logic is produced by using a multi-valued condition in the classical structure as shown below.





A program utilizing these control structures tends to have no statement labels. (The actual implementation of these structures in a non-structured language like FORTRAN will require the use of

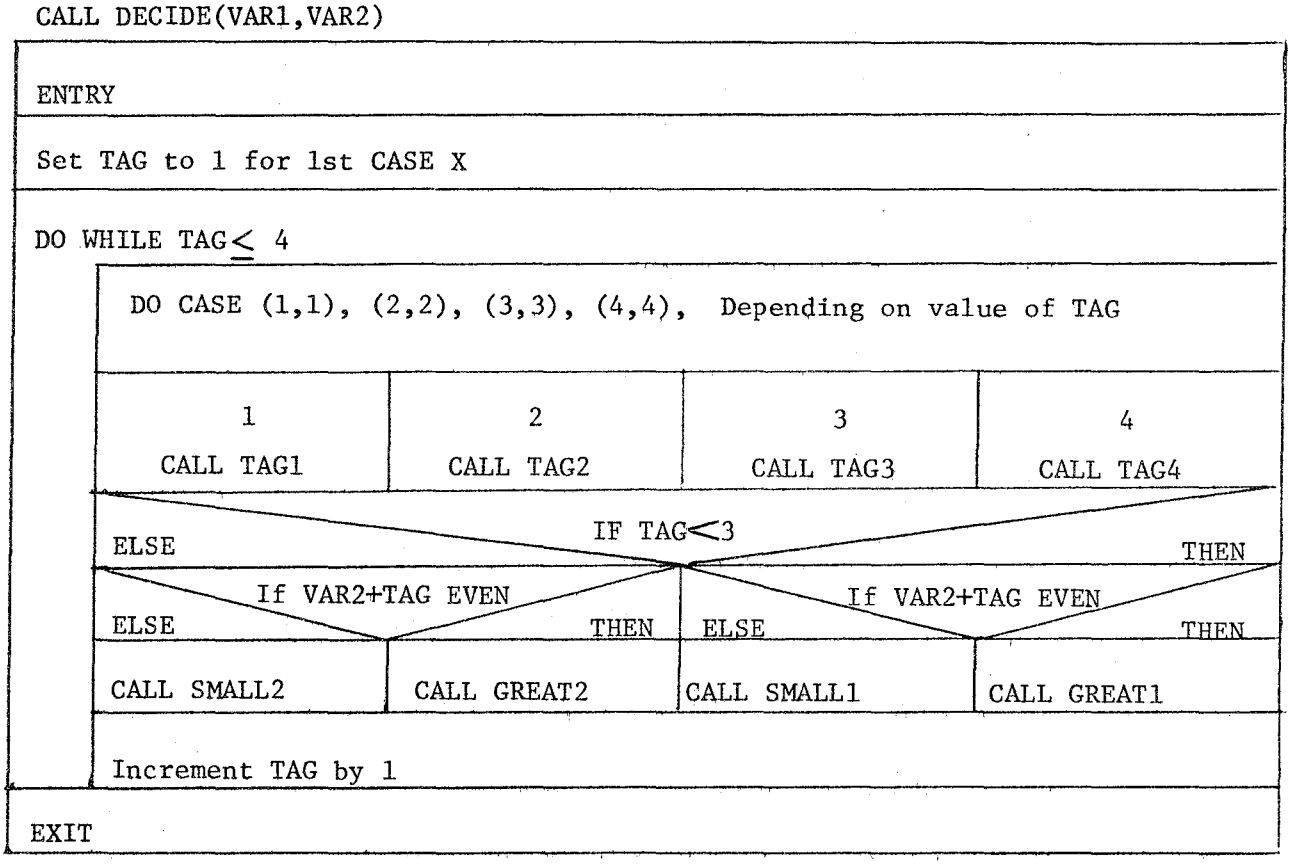
statement labels. See Section 1.3 - CODING.) Utilizing these control structures in a top-down design, a module will eliminate arbitrary and capricious branching and result in a more precise flow of data.

The hand drawn Chapin charts (Figure 5) will be generated in the design process for formal "walk through" reviews of the structured design. The purpose of the review will be to uncover flaws in the design. The Chapin chart will enable the reviewers to examine the entire logic of the module.

In addition to the use of the restricted control structures, the Chapin charts will also contain other attributes for ease of understanding. The chart will contain the title and name of the module. The module name should be descriptive of the function performed by the module and is the name that is used to reference the module (for example, in a CALL statement).

The language for the Chapin chart should not be cryptic to the point that only the designer understands the logic. Neither should it be so wordy that it can't fit in the box.

The use of the IF THEN/ELSE control structure will sometimes result in a do-nothing or NULL statement from the question. It is preferred that the NULL statement be designed and implemented from the ELSE path of the question.



Purpose - To initialize TAG areas and build tables.

Date - Designed/9/30/75, JD - Coded/10/15/75, MP

Functions - Call individual TAG areas in sequence. On the first two passes, build a small or large type one table first depending upon the value of the input parameter VAR1. On the next two passes, build a small or large type two table first depending upon the value of the input parameter VAR2.

Inputs - VAR1 = 1 if Great Table 1 is to be built first
VAR1 = 2 if Small Table 1 is to be built first
VAR2 = 1 if Great Table 2 is to be built first
VAR2 = 2 if Small Table 2 is to be built first

Outputs - None

Figure 5. Typical Chapin chart with external specifications.

Module lengths should be limited to a manageable size. No firm rule can exist for size; however, the tendency is to have between 10 and 100 lines of executable statements. With this size, a single entry, single exit, and no arbitrary jumps to other parts of the program, there is little need for page-turning or holding several places which must be referenced constantly.

The function performed by the module should be described in a single sentence followed by an expanded description, if necessary. The expanded description can be a narrative description, tables, etc., and should be easily adaptable to card format for inclusion in the programming documentation.

There should be a precise description of all input and output data for the module. It should include all parameters, any physical order, size, type, and range of valid values. A full description of module interconnections is necessary as it will usually affect any calling module.

Any external effects should be explained, e.g., the reading of a tape or printing.

The module name, functional description, input and output description, and external effects will be called the module's external specifications. For design, these items will be placed on the Chapin chart and/or additional pages if necessary. These items will be prepared to be carried onto the program listing as the first half of a module's prologue.

1.2.4.3 Pseudo Code

After the design "walk through" and approval, the Chapin chart will be converted into indented control structure pseudo code in punched card format. This will constitute the second half of the module's prologue.

Pseudo code, English phrases derived from the Chapin chart, describes the flow of the control structure. The simple sequence is a statement. The IF THEN/ELSE structure is divided into three parts; (1) IF is usually a one line question; (2) THEN is a statement to be executed if the answer is true; and (3) ELSE is a statement if the answer is false. The THEN statement and/or ELSE statement can be followed by other questions and/or statements. The DO UNTIL, SO WHILE, and CASE are statements.

The pseudo code acts as a bridge between the design and coding phases. It is a transformation of the highly graphic, parallel vision, Chapin charts into a form similar to the top-down, straight line, final source code. As such, there are several guidelines for converting the Chapin charts to pseudo code.

a. All decisions which alter the simple sequence flow of the program will be shown. If, during coding, a FORTRAN flow altering statement is introduced, it must be reflected in the pseudo code.

b. All FORTRAN CALL statements must be reflected as a simple sequence statement in pseudo code.

c. Each FORTRAN statement which is a simple sequence type does not require a matching statement in the pseudo code if it is part of a group of FORTRAN statements which performs a common pseudo code statement.

Example:

PSEUDO CODE	FORTRAN STATEMENT
Calculate X, Y, Z coordinates	X = _____
	Y = _____
	Z = _____

d. The control statements IF, DO WHILE, DO UNTIL, and CASE always denote additional statement(s) will follow. Each of these control statements will use an appropriate END statement to denote the end of a particular set. The END statement will be indented the same number of columns as its subject. The END statements are ENDIF, ENDDO, and ENDCASE.

e. The pseudo code will be written in a format with strict indentation in each group and subgroup of statements for ease of understanding and clarity. See Section 1.3.1 - Source Code Documentation for specific rules.

1.3 CODING

To ensure ease of understanding, maintaining, and interchanging of ANOPP code, certain standards will be imposed. These standards encompass both documentary comments and specific language statements. It is recommended that any exceptions to standards be employed only within the bounds of a specific module and not be allowed to couple with other modules.

1.3.1 Source Code Documentation

Comment statements in any programming language are both source code and documentation. Thus, various kinds of descriptive information which would normally appear in publishable programming documentation can be captured as comments in the source code also. For ANOPP, design and documentary information will be brought together and placed in the program source listing. This information will be contained in a special module prologue section at the beginning of a routine and in regular comment statements interspersed among the executable statements of a routine. The prologue section should explain the purpose and functioning of a routine as well as the flow of control within the routine. If any coding standard is violated, it must be noted in the prologue and, as an additional comment, in the executable statements. The in-line comments are supplementary in nature and should explain special cases or values and other non-obvious implementations.

The first line of a routine is the program header, be it PROGRAM, SUBROUTINE, FUNCTION, BLOCK DATA, or IDENT. The module prologue will appear immediately following the program header and preceding any other lines of source code. The source code and optional comments will follow the prologue.

The module prologue contains both descriptive information and a pseudo code translation of the module's Chapin chart. The definition of prologue contents and format is given below. An example of a prologue is illustrated in Figure 1.

Column	Statement
1	* (all prologue cards have an asterisk in column one)
5-9	BEGIN (begin prologue)
10	PURPOSE - narrative description of purpose
10	DATE - design-date-name, code-date-name, test-date-name
10	FUNCTION - narrative description of functions
10	INPUTS - input variables and settings
10	OUTPUTS - output variables and settings
10	DATA STRUCTURES - descriptions or references to files, common blocks, table, etc. used by this module
5-9	ENTRY - (begin pseudo code)
3-8	Statement Number
10,15,20,etc.	Pseudo Statement (simple sequences and the following sets of key words should be aligned in order within a set: IF, THEN, ELSE, ENDIF; DO CASE, CASE, ENDCASE; DO WHILE, DO UNTIL, ENDDO. Subsequent substructures should be indented 5 spaces.

```
      SUBROUTINE DECIDE (VAR1,VAR2)
*
*     BEGIN
*          PURPOSE - TO INITIALIZE TAG AREAS AND BUILD TABLES
*          DATE - DSGN JKD 10/12/75. CODE BPN 10/23/75. TEST
*          FUNCTIONS - CALL INDIVIDUAL TAG AREAS IN SEQUENCE.
*                      IF TAG IS LESS THAN 3 BUILD SMALL OR
*                      LARGE TABLE DEPENDING ON PARAMETER
*                      VAR1. IF TAG IS GREATER THAN 2 BUILD
*                      SECOND SET OF TABLES.
*          INPUTS - VAR1 = 1 BUILD GREAT TABLE 1
*                        = 2 BUILD SMALL TABLE 1
*                   VAR2 = 1 BUILD GREAT TABLE 2
*                        = 2 BUILD SMALL TABLE 2
*          OUTPUTS - NONE
*
*     ENTRY
*
*          SET TAG EQUAL 1
*       20 DO WHILE TAG LESS THAN OR EQUAL 4
*               DO CASE (1(40),(2,50),(3,60),(4,70) ON TAG
*       40           CASE 1     CALL TAG1
*       50           CASE 2     CALL TAG2
*       60           CASE 3     CALL TAG3
*       70           CASE 4     CALL TAG4
*       80      ENDCASE
*               IF TAG IS LESS THAN 3
*      100      THEN IF VAR1 * TAG IS EVEN
*      110           THEN CALL GREAT1
*      120           ELSE CALL SMALL1
*      130           ENDIF
*      140      ELSE IF VAR2 * TAG IS EVEN
*      150           THEN CALL GREAT2
*      160           ELSE CALL SMALL2
*      170           ENDIF
*      180      ENDIF
*      190      INCREMENT TAG BY 1
*      200 ENDDO
*     EXIT
*
*     END
*
```

Figure 1. Prologue and executable statement listing.

```
      INTEGER TAG,VAR1,VAR2
   20 DO 190 TAG=1,4
      GO TO(40,50,60,70)TAG
   40 CALL TAG1
      GO TO 80
   50 CALL TAG2
      GO TO 80
   60 CALL TAG3
      GO TO 80
   70 CALL TAG4
   80 CONTINUE
      IF(TAG-3) 100,140,140
  100 IF (.NOT.((((VAR1+TAG)/2)*2-(VAR1+TAG)) .EQ.0)) GO TO 120
  110 CALL GREAT1
      GO TO 130
  120 CALL SMALL1
  130 CONTINUE
      GO TO 180
  140 IF (.NOT.((((VAR2+TAG)/2)*2-(VAR2+TAG)) .EQ.0)) GO TO 160
  150 CALL GREAT2
      GO TO 170
  160 CALL SMALL2
  170 CONTINUE
  180 CONTINUE
  190 CONTINUE
  200 CONTINUE
      RETURN
      END
```

Figure 1. Prologue and executable statement listing. (Continued)

5-8 EXIT (end pseudo code)

5-7 END (end prologue)

The statements in the pseudo code should be labeled with statement numbers where appropriate. These numbers should be in numerically ascending sequence from the top down. Corresponding FORTRAN statements in the source code should be similarly numbered in the same top-down sequence.

Section 1.2.4.3 described the design conversion from Chapin chart to pseudo code. Section 1.3.2 will describe the translation from pseudo code to FORTRAN source code. Thus, the pseudo code in the prologue is a highly visible bridge between the module as designed and the module as coded. Capturing this documentation in the source code will simplify the tasks of understanding, testing, and maintaining a module's code.

1.3.2 FORTRAN Language Standards

The requirements of ANOPP will impose certain restrictions on the use of the normal FORTRAN language for two reasons. First, the requirement for machine independence demands the use of a FORTRAN subset that operates compatibly on several manufacturer's computers. Second, the set of requirements for structured programming and its attendant simple logic structures demand several implementation algorithms since FORTRAN is not a structured programming language.

1.3.2.1 Machine Independence

FORTRAN, a high level compiler language, is relatively machine-independent. Even so, standard FORTRAN (ANSI X3.9-1966) has not been implemented by the same or different manufacturers to be completely independent of machine architecture. However, a fundamental precept of ANOPP development is to minimize implementation and conversion problems on the major third generation scientific computers (CDC CYBER series, IBM 360/370 series, UNIVAC 1100 series). To this end, ANOPP code will conform to ANSI standards as defined in the FORTRAN Extended Version 4 Reference Manual (Control Data Publication Number 60305601) subject to the restrictions listed below.

NON-ANSI constructions (indicated by shaded areas in the reference manual) must not be employed. The following are standards that are to be followed to maximize machine independence.

1. The magnitude of an integer constant or variable may not be greater than $2^{31} - 1$.
2. Subscripted variables should contain no more that 3 subscripts.
3. Array variables must be referenced with explicit subscripts, e.g., A(1) = 0, not A = 0.
4. A CONTINUE statement requires a FORTRAN statement number.
5. The PAUSE statement is not to be used.
6. The NAMELIST statement is not to be used.
7. Implied DO's in DATA statements are not allowed.

8. The last statement of a DO loop may not be a logical IF statement.

9. BLOCK DATA subprograms may contain only type (e.g., REAL, INTEGER), DIMENSION, COMMON, and DATA statements.

10. All variables containing Hollerith data should be limited to eight characters left-justified and blank-filled. The forms nL and nR should not be used for Hollerith data. The form nH should be used. When using A(i) format specification, i must not exceed 8. A(3), A(6), A(8) are valid, A(10) is invalid.

11. Packed fields within a computer word should not be used.

12. Octal (O or B) or Hex (Z) in DATA or FORMAT statements may not be used.

13. Specification statements should precede any executable statement.

14. The order of specification statements should be as follows:

    COMPLEX

    DOUBLE PRECISION

    REAL

    INTEGER

    LOGICAL

    EXTERNAL

    DIMENSION

    COMMON

    EQUIVALENCE

DATA

15. The variables in a COMMON block should be ordered as follows: complex, double precision, real, integer, and logical.

16. Variables stored as single precision cannot be referenced as double precision variables (via the FORTRAN EQUIVALENCE statement) because of the different internal word storage format for single and double precision words.

17. Caution must be exercised to insure that types (REAL, INTEGER, etc.) of FORTRAN functions agree in the function subprogram and in the calling program. This agreement between types is necessary for machines (e.g., IBM 360) in which REAL and INTEGER values of FORTRAN functions are returned in different registers.

18. No attempt to extend the length of arrays through the EQUIVALENCE statements should be made.

19. Caution must be exercised when using the EQUIVALENCE statement. Optimizing compilers do not guarantee that the values used for the equivalenced variables will be the expected value. Hence, EQUIVALENCE should be used only between variables which have non-intersecting use spans in a program. Storage and retrieval of a variable value is not necessarily in the order given by FORTRAN source.

20. Multiple entry routines and routines with nonstandard returns are not to be used.

21. There must be agreement with respect to the number of arguments and the type of each argument in the argument list of a calling program and the called subroutine.

22. Only the carriage control character "1" may be used to control printer paging. No spacing or suppression of spacing characters may be used.

23. Modification of the length of an explicit type declaration (e.g., REAL*8) is not allowed.

24. Deck (or member) names for subroutines should be six or less characters and should agree with the primary entry point names. Deck names for Block Data subprograms should end with the characters "BD".

25. FUNCTION subprograms whose type is not implicit must be typed in the FUNCTION statement. For example, use

```
DOUBLE PRECISION FUNCTION ABC(X)
```

and not

```
FUNCTION ABC(X)
DOUBLE PRECISION ABC
```

26. The name of a FUNCTION subprogram must appear somewhere within the subprogram.

27. All subscripted variables appearing in EQUIVALENCE statements must be subscripted, e.g. use EQUIVALENCE (A(1), X(1)) instead of EQUIVALENCE (A,X).

28. DO loop indices may not be greater than $2^{17} - 1$ (131,071).

29. Logical operations are permitted on non-logical variables only

using supplied functions IAND, IOR, ICOMPL, IXOR.

30. Subscripts may not contain subscripted variables.

31. Actual subroutine parameters that are changed by the called subroutine must have unique locations.

    Example: CALL SUB(A,A) where SUB is as follows:

```
SUBROUTINE SUB(C,D)
C = 10
RETURN
END
```

    is not allowed.

32. No DATA statements for variables in common blocks outside BLOCK DATA programs will be used.

33. Blank common will not be used.

34. ENCODE, DECODE, or similar installation or machine dependent routines will not be used.

35. Branching into the range of a DO statement is not allowed.

36. It is preferred that the use of constant numbers for referencing or indexing tables be restricted.

1.3.2.2 Structured FORTRAN

FORTRAN is not a structured programming language. FORTRAN syntax does not directly include the logic constructs defined in Section 1.2.3 and Section 1.2.4 of this document. However, several implementation algorithms can be defined to permit adherence to the concept of structured programming.

1.3.2.2.1 Simple Sequence

FORTRAN syntax permits easy implementation of the simple sequence structure. There is no need for statement numbers within the sequence except for format statements that do not alter the flow of execution. Entry to the sequence may require a statement number if it is entered as the result of a previous branching structure.

Example:

Pseudo Code	FORTRAN Code
GET X	READ 100, X
COMPUTE SQUARE ROOT OF X	SX = SQRT(X)
PUT RESULT	PRINT 100, SX

1.3.2.2.2 IF THEN/ELSE

FORTRAN syntax does not include an IF THEN/ELSE structure. However, various combinations of Arithmetic If, Logical If, Go To, and Continue statements can provide the two-branch logic desired. The rules for their use are as follows:

1. The THEN path must precede the ELSE path in top-down order in both the pseudo code and the FORTRAN code.
2. The ENDIF statement must be represented by a numbered Continue statement.
3. The Arithmetic If statement with two of the three branches equal is the preferred implementation.
4. The Logical If must be used with logical variables or multiple

relational expressions.

5. The Logical If must be implemented with a .NOT. condition and a Go To the ELSE path.

Example 1: Arithmetic If

Pseudo Code	FORTRAN Code
IF ANGLE .LE. 180	IF (PI - THETA) 2, 1, 1 (THETA - PI) 1, 1, 2
1 THEN COMPUTE SIN(ANGLE)	1 SINTH = SIN(THETA) GO TO 3
2 ELSE COMPUTE - SIN(180 - ANGLE)	2 SINTH = -SIN(PI - THETA)
3 ENDIF	3 CONTINUE

Example 2: Logical If

Pseudo Code	FORTRAN Code
IF SWITCH is TRUE	IF(.NOT.(SWITCH)) GO TO 1
THEN CALL SUBA	CALL SUBA GO TO 2
1 ELSE CALL SUBC	1 CALL SUBC
2 ENDIF	2 CONTINUE

Example 3: ELSE path null

Pseudo Code	FORTRAN Code
IF X .LT. ZERO	IF (X) 1,2,2 (preferred) (.NOT.(X .LT. 0)) GO TO 2
1 THEN X = ABS(X)	1 X = ABS(X)
ELSE NULL	
2 ENDIF	2 CONTINUE

1.3.2.2.3 CASE

DO CASE (condition 1, statement 1) .....(condition m, statement n) on test

variable

The DO CASE structure can be implemented with a variety of programming techniques employing a conbination of Go To, Computed Go To, Logical If, Arithmetic If, and Continue statements. In all cases the ENDCASE statement must be represented by a CONTINUE statement with a statement number. Four basic examples are illustrated.

Example 1: Integer Test Variable

Pseudo Code	FORTRAN Code
DO CASE (1,1) (2,2) (3,3) (4,4) (5,5) on I	GO TO (1, 2, 3, 4, 5) I
1 Process Control Statement 1	1 CALL PRCS1 GO TO 6
2 Process Control Statement 2	2 CALL PRCS2 GO TO 6
3 Process Control Statement 3	3 CALL PRCS3 GO TO 6
4 Process Control Statement 4	4 CALL PRCS4 GO TO 6
5 Process Control Statement 5	5 CALL PRCS5
6 ENDCASE	6 CONTINUE

Example 2: Arithmetic If

Pseudo Code	FORTRAN Code
DO CASE (-,1) (0,2) (+,3) on X	IF (X) 1, 2, 3
1 SX = SQRT(-X)	1 SX = SQRT(-X) GO TO 4
2 SX = 0	2 SX = 0 GO TO 4
3 SX = SQRT(X)	3 SX = SQRT(X)
4 ENDCASE	4 CONTINUE

Example 3 - Logical If with an executable statement

Pseudo Code	FORTRAN Code
DO CASE ("A",1) ("B",2) ("C",3) ("D",4) on NAME	1 IF (NAME .EQ. 1HA) CALL SUBA
1 CALL SUBA	2 IF (NAME .EQ. 1HB) CALL SUBB
2 CALL SUBB	3 IF (NAME .EQ. 1HC) CALL SUBC
3 CALL SUBC	4 IF (NAME .EQ. 1HD) CALL SUBD
4 CALL SUBD	5 CONTINUE
5 ENDCASE	

Example 4 - Logical If with Go To

Pseudo Code	FORTRAN Code
CASE ("A",1) ("B",2) ("C",3) ("D",4) on NAME	IF (NAME .EQ. 1HA) GO TO 1
1 CALL SUBA	IF (NAME .EQ. 1HB) GO TO 2
2 CALL SUBB	IF (NAME .EQ. 1HC) GO TO 3
3 CALL SUBC	IF (NAME .EQ. 1HD) GO TO 4
4 CALL SUBD	GO TO 5
5 ENDCASE	1 CALL SUBA
	GO TO 5
	2 CALL SUBB
	GO TO 5
	3 CALL SUBC
	GO TO 5
	4 CALL SUBD
	5 CONTINUE

1.3.2.2.4 DO WHILE (condition) and DO UNTIL (condition)

The DO WHILE structure implies that condition testing is done before the sequence of operations is performed. The DO UNTIL structure implies that the sequence of operations is performed at least once before condition testing is done. These structures can be implemented in FORTRAN with various combinations of Arithmetic If, Logical If, Go To, Continue, and Do statements. The possible variations are too myriad to enumerate specifically. However, the following standards will promote adherence to the spirit of top-down structured programming.

1. A Do statement can be used for a DO UNTIL with an incrementing condition.
2. If a Do statement is used for a DO WHILE, then the condition must be tested as the first statement inside the Do or as the statement immediately preceding the Do if it is an incrementing condition with variables instead of constants.
3. All FORTRAN Do loops must end with a numbered CONTINUE statement.

1.3.2.2.5 CONTINUE

A CONTINUE statement is used if required to insure that the DO WHILE or DO UNTIL will stand alone (i.e., not depend on the previous or next pseudo code structure). A labeled CONTINUE statement can therefore appear in the FORTRAN code with a label number not appearing in the Pseudo code. The label numbers should, of course, appear sequentially in the FORTRAN code. Examples 1, 3, and 4 show a labeled CONTINUE not

shown in the Pseudo code but required for implementation. Example 2 shows the absence of any CONTINUE statement.

Example 1

Pseudo Code	FORTRAN Code
DO UNTIL (TABLE(I)=NAME FOR I=1 to TABLELENGTH)	DO 9 I=1, LTAB
.....	.....
.....	.....
10 ENDDO	IF (TABLE(I) .EQ. NAME) GO TO 10
	9 CONTINUE
	10 CONTINUE

Example 2

Pseudo Code	FORTRAN Code
10 DO UNTIL A EQUALS B	10 .....
.....	.....
ENDDO	IF (N .NE. B) GO TO 10

Example 3

Pseudo Code	FORTRAN Code
DO WHILE (I .LE. K FOR I = J TO K)	IF (J .GT. K) GO TO 10
.....	DO 9 I=J,K
.....	.....
10 ENDDO	9 CONTINUE
	10 CONTINUE

Example 4

Pseudo Code	FORTRAN Code
DO WHILE (A EQUALS B OR C GREATER THEN D)	IF (.NOT.(A.EQ.B .OR. C.GT.D)
.....	*GO TO 10
	.....
10 ENDDO	GO TO 9
	10 CONTINUE

1.3.3 Assembly Language Standards

There is no unique assembly language that is compatible across several manufacturers' computers. The assembly language standards for ANOPP, then, fall into two categories: (1) general requests to adhere to the spirit of structured programming and (2) specific interface requirements between FORTRAN and assembly language subroutines for a given machine.

1.3.3.1 General Rules

The following general rules will promote clarity and understanding while adhering to the spirit of structured programming.

1. Each routine shall have a module prologue as described in Section 1.3.1.
2. Multiple entry points and non-standard returns are not allowed.
3. Self-modifying code is not permitted. Instructions can change data only, they cannot change other instructions.
4. Assembly code must follow the same top down order as the pseudo code.

5. Liberal use of comments is recommended for clarity and understanding.
6. The prologue's pseudo code should be repeated in appropriate comment fields of assembly statements.
7. Local macro definitions should be found after the prologue at the beginning of a routine.
8. System macroes should briefly be explained and a document reference cited.

1.3.3.2 COMPASS/FORTRAN Interface

Several conventions must be observed when FORTRAN and COMPASS subroutines are intermixed. For FORTRAN Extended, the conventions are explained in the Reference Manual (CDC Pub. No. 60305600). A brief list follows:

1. Every COMPASS subroutine shall have:
   a. IDENT and END cards beginning in column 11,
   b. A trace word of the form VFD 42/name, 18/entry address,
   c. An entry point of the form name DATA 0,
   d. An entry point name agreeing with the deck name on the IDENT statement,
   e. Register A0 saved on entry and restored upon exit.
2. Function subroutines shall return single precision values in register X6; double precision and complex values are returned in registers X6 and X7.
3. Subroutine and function calls are performed by a return jump sequence with trace information and argument addresses passed

through an argument list. The form is as follows:

```
        SA1     ARGLIST
    +   RJ      =X external subprogram name
    -   VFD     12/line number, 18/trace word address
        ...
ARGLIST VFD     60/ARG1 address
        VFD     60/ARG2 address
        ...
        VFD     60/ARGM address
        VFD     60/0 end of argument address list
```

```
          IDENT SAMPLE

***
*
*P
* R
*  O
*   L
*    O
*     G
*      U
*       E
*
***

          ENTRY SAMPLE
TRACE     VFD   42/0LSAMPLE,18/SAMPLE      Trace word
TEMPAO    DATA  0                          Holding location for AO
EXIT      SA1   TEMPAO                     Restore AO
          SAO   X1
SAMPLE    DATA  0                          Entry point
          SX6   AO                         Save AO
          SA6   TEMPAO
          SAO   A1                         Save input argument list
          ...                              address
          ...
          SA1   ALIST                      Call SUB(A,B,C)
   +      RJ    =XSUB
   -      VFD   12/*-TRACE,18/TRACE
          ...
          SA1   AO+1                       Fetch 2nd argument
          SA1   X1
          ...
          ...
          EQ    EXIT       Restore AO and return through entry point
          ...
ALIST     VFD   60/A       Address of A
          VFD   60/B       Address of B
          VFD   60/C       Address of C
          VFD   60/0       End of argument list
   A      DATA  0          Storage for A
   B      DATA  0          Storage for B
   C      DATA  0          Storage for C
          ...
          END
```

Figure 2. Sample parts of COMPASS routine.

1.4 TESTS

Testing is the activity that takes coded pseudo and FORTRAN statements and removes compiler statement errors, input formatting errors, output formatting errors, and program structural and logic errors. The testing activity is composed of (1) desk checking, (2) component testing, (3) integration testing, and (4) system testing.

1.4.1 Desk Checking

Upon completion of coding of the FORTRAN or assembly statements for a module, the product should be reviewed with the Chapin Chart and the pseudo code for completeness and accuracy. The module is compiled and all compiler generated errors are removed to obtain an error-free compilation.

1.4.2 Component Testing

Component or isolation testing is that activity which takes a module, exercises it through its full range of inputs and outputs, and evaluates its performance for any necessary correction. Each and every path of a module must be exercised during component testing. Stubs for other modules that are referenced must be generated to allow a smooth run to completion.

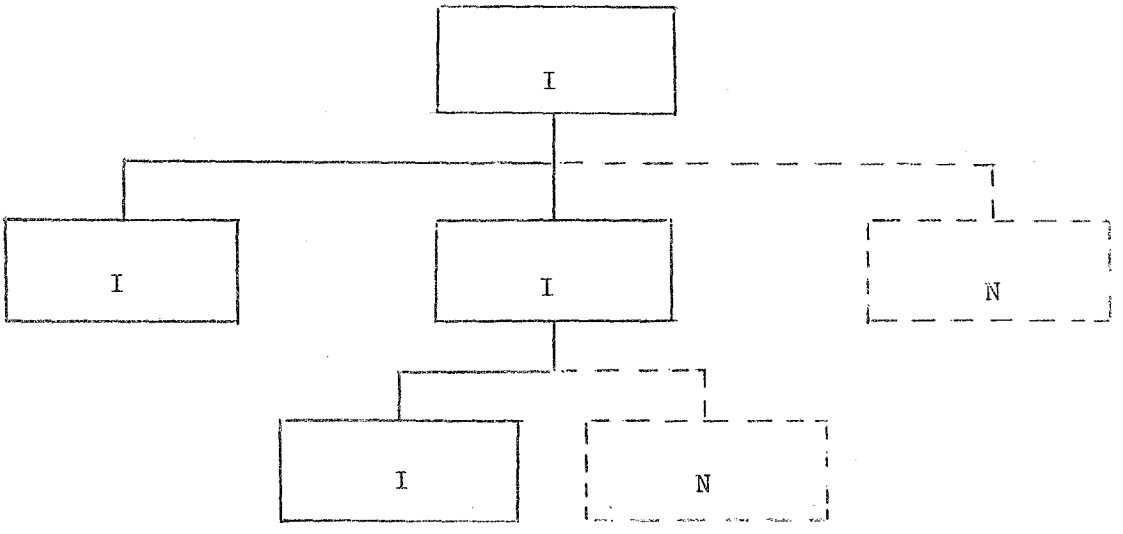
Standards for component testing will produce tests that will:

1. exercise typical error free cases,
2. exercise error free worst case,

3. produce each error code,
4. produce variations of errors,
5. vary all system parameters affecting the module for the above runs, and
6. vary user options affecting the module for the above runs.

1.4.3 Integration Testing

After component testing, the module is integrated into the program. In the figure below, all modules designated with I are integrated and the modules designated with N are not integrated. A module is never integrated into the program unless it is subordinate to a previously integrated module.



When a module is integrated into the program, integration tests will be performed. Integration and testing is the activity which places the tested module into the program and exercises the module.

The module is exercised as thoroughly as possible for interaction with other modules. The tests should check for:

1. typical case with no errors,
2. worst case with no errors,
   a. efficiency
   b. any designed limits
3. each error code,
4. all possible errors in one entry,
5. various typical cases,
6. various system parameters that affect the module, and
7. various user options that affect the module.

The test cases and results of integration tests will be documented and saved for later use. These test cases can be used to determine if the program is operating as designed.

1.4.4 System Tests

System testing is the activity of exercising the program utilizing all inputs in various combinations. According to the concepts of structured programming and top-down module integration, as the last module is integrated and tested, testing of the entire program will be complete. However, tests will be conducted for:

1. the ANOPP control statement stream with no errors and composed of
   a. simple sequences,

b. various typical combinations, and

c. worst case combinations;

2. the ANOPP control statement stream with errors, such as,

a. meaningless input,

b. no input,

c. stacked sets of input, and

d. errors;

3. all system parameters for

a. typical settings,

b. special cases,

c. worst cases, and

d. illegal values.

1.5 PUBLISHABLE DOCUMENTATION

1.5.1 Types of Publishable Documentation

A program of ANOPP's magnitude requires clear and complete documentation to be of value to users and programmers. Engineers and users must understand the available prediction capabilities and know how to formulate a problem and obtain a solution. Programmers must know how to install, modify, and add to the system. Such documentation will be provided in four separate, indexed, stand-alone documents:

1. Programmers's Manual,
2. Theoretical Manual,
3. User's Manual, and
4. Demonstration Problem Manual.

1.5.1.1 Programmer's Manual

The Programmer's Manual will contain all coding information and specifications for the Aircraft Noise Prediction Program. It will be written for use by programmers to install, execute, modify, and add modules to the program. As such, it will contain:

1. a detailed introduction that will describe the concepts and functions of ANOPP;
2. the standards for design, coding, testing and program documentation to insure compatibility and ease of maintenance;

3. description of data and tables;
4. executive, data management, utility, and functional module descriptions;
5. instructions for installation and operation;
6. instructions for modification and addition of modules to the system;
7. support program descriptions; and
8. easily updated index.

1.5.1.2 Theoretical Manual

The Theoretical (or methods) Manual will provide a concise mathematical description of the methods employed in the computational or functional modules. It will describe the analytical or empirical methods and will outline the methods of solution, including all implicit and explicit assumptions, limits of use and limits of accuracy. References to published material should be included in the text and the index. A user can refer to this manual to determine the engineering and mathematical methods that are available in the program.

1.5.1.3 User's Manual

The User's Manual will be structured to accommodate the needs of different levels of users. A user will employ this manual to formulate problems and anticipate results. The manual will provide instructions and descriptions for the preparation of problem data and explain how

to invoke the various options provided for problem solution. The User's Manual will thoroughly explain the Executive Control Language and general related capabilities.

1.5.1.4 Demonstration Problem Manual

The Demonstration Problem Manual will contain detailed descriptions of sample problem input and solutions. This manual will be utilized for user education and system validation. It will be beneficial to the engineer user and his programming staff.

1.5.2 Publishable Manual Preparation

1.5.3.1 General

ANOPP documentation will be typed on 11" x 14" mats that will be supplied by ANOPO. These mats will be reduced to 80 percent of their original size during the printing process.

Documents will be prepared using tape cassettes compatible with the equipment available to ANOPO. The equipment available to ANOPO is an IBM MAG Card II Typewriter, System Model No. 6616; specifications: 2 cassette dual pitch word processing system.

The cassette tapes of the ANOPP manuals will be furnished to ANOPO with an index to facilitate cataloging and filing the tapes.

Computer printout used in the manuals must be clear and sharp. To ensure this, the unlined side of the computer paper should be used

and a new ribbon should be inserted on the printer.

To change camera-ready manuscripts, correction tape is preferred over mortising (i.e., cutting and pasting). Also, for one-letter corrections, a chalk-like substance may be used. ERASURES AND OPAQUE WHITE CORRECTION FLUID ARE NOT ACCEPTABLE.

Minor modifications to pages of published ANOPP documentation will be communicated to ANOPO by means of the Documentation Change Report (DCR).

The general format of the manuals will be:

1. Front Cover
2. Inside Cover Page
3. Preface
4. Table of Contents
5. Page Status Log
6. Text Body
7. References
8. Index
9. Back Cover

1.5.2.2 Spacing

Double-spacing will be used except where groups of a few single-spaced lines separated by double-spacing for the groups is more desirable for clarity or appearance. An example of the exception could be DATA BLOCK DESCRIPTIONS.

Paragraphs will be indented 5 spaces and will be separated from each other by 2-1/2 lines. A line associated with an unnumbered, underlined explanatory heading will be indented 5 spaces.

Section and subsection titles will be separated from the text (above and below the title and from each other) by 3 lines.

1.5.2.3 Section Numbering

Major sections, i.e., those with one number, will be typed with all uppercase letters and will be identified with a decimal classification, i.e., one number followed by a period, as follows:

12. DOCUMENTATION

Major subsections, i.e., those with two numbers, will be typed with all uppercase letters and will be identified with a decimal classification and two numbers, as follows:

12.2 MANUAL PREPARATION

Minor subsections, i.e., those with three numbers, will be typed with initial capitals, underlined, and identified with a decimal classification and three numbers, as follows:

12.2.5 Contents of Manual

Further subdivision of minor subsections will be typed with initial capitals, will not be underlined, and will be identified with a decimal classification and four or more numbers, as follows:

12.2.5.3 Text Printing

1.5.2.4 Page Numbering and Running Headings

Major subsections will begin at the top of an odd-numbered page. (Odd-numbered pages will be printed on the right and even-numbered pages will be printed on the left.) Module descriptions should begin at the top of an odd-numbered page. Other units such as Data and Table Descriptions should begin at the top of a page where clarity or convenience of use is thereby improved. In the case of large major subsections, minor subsections may begin at the top to the next page.

Page numbers will be centered at the bottom of each page. The number will indicate the major subsection identifier and page number within the subsection, separated by a hyphen. Examples are:

6.1-1

6.1-2

6.1-3

The numbers of pages changed at a later date will use the format of major subsection identifier, hyphen, page number followed by the date of the change (mm/dd/yy), as follows:

6.1-1	Original page
6.1-2 (12/09/75)	Changed page

Pages inserted at a later date will be identified by the major subsection identifier, hyphen, page number, decimal and number followed by the date of the insertion in the form (mm/dd/yy), as follows:

6.1-1	Original page
6.1-2 (12/09/75)	Changed page
6.1-2.1 (12/09/75)	Added page
6.1-2.2 (12/09/75)	Added page
6.1-3	Original page

If an odd number of pages is to be inserted, one blank page with a running header and a page number should be added to ensure consistency. Such a blank page will contain the following sentence: THIS PAGE HAS BEEN LEFT BLANK INTENTIONALLY.

Pages inserted at a later date between pages of insertions made subsequent to the original issue will be identified by the major subsection identification number, hyphen, page number, decimal, added page, decimal, inserted page and date as follows:

6.1-1	Original page
6.1-2 (12/09/75)	Changed page
6.1-2.1 (12/09/75)	Added page
6.1-2.1.1 (01/10/76)	Inserted page
6.1-2.1.2 (01/10/76)	Inserted page
6.1-2.2 (12/09/75)	Added page
6.1-3	Original page

Running headers, in capitals, will be centered at the top of each page. The major section name will be used as the running header on EVEN-NUMBERED PAGES, and the major subsection name will be used as the running header on ODD-NUMBERED PAGES. There is one major exception

to this rule: for the first page in every major subsection, the major section name will be used as the running header. Care must be exercised in determining running headers for pages to be inserted. For example, the page to be inserted between 6.1-3 and 6.1-4 is 6.1-3.1 and is considered to be an even-numbered page for the purpose of determining the running header. The reason for this is that 6.1-3.1, being printed on the back of 6.1-3 (an odd-numbered page) is in effect an even-numbered page. A blank "odd-numbered" page, 6.1-3.2, with subsection running header and page number must be typed to ensure consistency. This blank page will contain the following sentence: THIS PAGE HAS BEEN LEFT BLANK INTENTIONALLY.

1.5.2.5 Equations

Equations will be numbered consecutively beginning with 1 for the first equation in each major subsection. References to equations outside a major subsection must refer to both the equation number and the major subsection number, e.g., See Section 5.6, Equation 12. If the reference is to another manual, the manual name (Theoretical Manual, User's Manual, Programmer's Manual) will be given, e.g., See ANOPP Theoretical Manual, Section 5.6, Equation 12.

Equations will be centered on the line and separated from the text by three blank lines. Equations will be punctuated as part of the text and will be identified at the righthand margin with its Arabic numeral equation number in parenthesis.

When a group of equations appears in succession without text between them, the longest equation will be centered and the equal signs of the remaining equations will be aligned with the equal sign of the longest equation.

The transpose operator for a matrix should be placed outside the brackets (i.e., $[A]^T$ is correct; $[A^T]$ is incorrect). The same rule applies to the inverse operation (i.e., $[A]^{-1}$ is correct; $[A^{-1}]$ is incorrect).

All subscripts for matrices will be lowercase letters (e.g., $[K_{99}]$, $[K_{fs}]$).

All plus and minus signs in equations will be preceded and followed by one space. There will be two spaces before and after all equal signs in equations. There will be no spaces between parenthetical expressions. For example: $(A)(B) + (D) = C$

Equations inserted at a later date will be numbered as decimal parts of the preceding equation. For example, two equations inserted between Equation 5 and Equation 6 will be designated by Equation 5.1 and Equation 5.2, respectively. For more complicated cases, follow the rules given in Section 1.5.2.4 of this document.

1.5.2.6 Tables, Figures, and References

Tables and figures will be numbered consecutively beginning with the first table or figure in each major subsection. References to

tables and figures outside a major subsection must refer to both the table or figure number and the major subsection number, e.g., See Section 5.6, Table 2. If the reference is to another manual, the manual name (ANOPP Theoretical Manual, ANOPP User's Manual, ANOPP Programmer's Manual) will be given, e.g., See ANOPP User's Manual, Section 5.2, Table 2.

Table titles will be typed with initial capitals. Periods will follow the Arabic table number and the end of the complete title as follows:

Table 3. This Is an Example of a Table Title.

Figure captions will be typed in lower case letters except for the first letter of the first word. Periods will follow the Arabic figure number and the end of the complete caption as follows:

Figure 4. This is an example of a figure caption.

Single line captions will be centered under the figure, and multiple-line captions will be left-justified with the last line centered.

References will be listed at the end of each major section and will be numbered consecutively beginning with 1 for the first reference in each major section.

Tables, figures, and references inserted at a later date will be designated by a number followed by a decimal and number. For example, two figures inserted between Figure 2 and Figure 3 will be designated Figure 2.1 and Figure 2.2, respectively.

The words Equation, Figure, Reference, Section, and Table will be spelled out with initial capitals when used either in the text or in a caption. The associated Arabic numeral will not be enclosed in parentheses.

1.5.2.7 Capitalization

Data block names, module names, Data card names, entry point names, and FORTRAN variable names will all be capitalized and the letter O will be slashed (Ø).

Care should be exercised in the use of initial capitals. Formal type names should be capitalized throughout the manuals.

1.5.2.8 Punctuation

Commas will be used to separate the elements in a series and a comma will be placed before the final conjunction.

For punctuation of potential executable statements, punctuation characters should be representative of the actual coded statement.

1.5.3 Changes to Baseline Manuals

The four ANOPP MANUALS (Theoretical Manual, User's Manual, Programmer's Manual, and Demonstration Problem Manual) when delivered to ANOPO via paper, called mats, and IBM MAG Card II compatible cards constitute baseline documents. When information in a baseline document is added, deleted, or changed, a formal written update to the baseline

document is required.

To initiate a change to a baseline document, a Documentation Change Report (DCR) is required and must be submitted to the maintenance organization. A DCR is shown in Figure 1.

The report will be reviewed by the maintenance organization and/or ANOPO for appropriateness and extent of change. Changes to manuals can affect software. The results of the reviews will consist of comments, required changes, and suggested changes as well as rejection or acceptance of the change. If the change is unacceptable, the submitter will be so informed and told what action must be taken prior to resubmission. If the change is acceptable, the maintenance organization will be informed so the change can be incorporated into existing mats and cassettes. If the change affects software, a Software Change Report (SCR) must be submitted with the DCR. If the change affects more than one manual, those manuals and page numbers must be indicated in the COMMENTS of the DCR.

All changes will be rigidly controlled, reviewed, cataloged, accounted, and filed. Documentation Page Status Logs (DPSL) will be maintained for and in each manual. A Documentation Change Report Status Log will be maintained with the changes for each manual. If a change affects more than one manual, it will be checked on the primary Documentation Change Request Report Status Log by the DCR number.

DCR No. _____

DCR No. _____

ANOPP DOCUMENTATION CHANGE REPORT (DCR)

Originator: _____ Date: _____

Organization: _____ Phone No.: _____

Manual		Page Numbers
	Theoretical	
	User's	
	Programmer's	
	Demonstration	

Description and reason of change:

(Attach a copy of the page(s) to be changed with corrections typed. Use separate pages if necessary.)

Comments

Editor Approval	ANOPP Approval	DPSL Entry	Change Made	ANOPO Verif.	Editor Verif.	DPSL Entry
Date	Date	Date	Date	Date	Date	Date

Figure 1. ANOPP DOCUMENTATION CHANGE REPORT (DCR)